Résolution efficace du consensus généralisé dans les systèmes répartis par passage de messages

Pierre Sutra[1], Marc Shapiro[1]

[1] *INRIA Paris-Rocquencourt et LIP6, France*

La construction d'un objet partagé entre plusieurs processus nécessite la résolution du consensus. De ce fait, la résolution du consensus est un sujet central et récurrent dans le domaine des systèmes répartis. Dans un article récent, L. Lamport suggère de prendre en compte la commutativité des accès concurrents à un objet partagé afin d'améliorer la vitesse de résolution du consensus. Cet article introduit notamment Generalized Paxos, un algorithme qui résout le consensus dans les systèmes répartis par passage de messages en deux étapes de communication, si les accès concurrents sont commutatifs ou spontanément ordonnés par le réseau. Toutefois, si une collision se produit, c'est à dire que deux accès concurrents et non-commutatifs ont lieu, Generalized Paxos exécute quatre étapes de communication supplémentaires. L'algorithme devient alors moins performant que les solutions antérieures. Dans cet article nous décrivons une nouvelle solution au consensus généralisé : *FGGC*. *FGGC* est optimal au regard des fautes et de la vitesse d'exécution car : (i) pour tolérer f pannes franches, *FGGC* utilise $2f+1$ processus, (ii) si toutes les commandes sont commutatives, ou spontanément ordonnées par le réseau, *FGGC* résout le consensus en deux étapes de communication, (iii) les collisions sont résolues en une seule étape de communication, et (iv) *FGGC* utilise seulement $f+1$ processus pour progresser.

Keywords: système réparti par passage de messages, consensus, Paxos.

1 Introduction

Un calcul réparti requiert usuellement l'emploi d'un objet partagé, c'est à dire un objet accédé de manière concurrente par plusieurs processus. Deux propriétés sont souhaitables [Her91] : (*linéarisabilité*) l'objet se comporte comme s'il était accédé séquentiellement par un seul processus, (*sans-attente*) tout accès à l'objet partagé termine en un nombre fini d'étapes de calcul. Le partage sans-attente et linéarisable d'un objet quelconque nécessite de résoudre le consensus, c'est à dire nécessite que les processus ordonnent les accès concurrents. Ainsi, lorsqu'un objet partagé est accédé fréquemment par les processus, la vitesse de résolution du consensus détermine la vitesse d'exécution du système dans son ensemble.

Afin d'augmenter la vitesse du consensus, L. Lamport observe dans un article récent [Lam05] que les accès concurrents commutatifs ne nécessitent pas d'être ordonnées. En d'autres termes, les processus peuvent ne pas être d'accord sur l'ordre dans lequel l'objet partagé est accédé tant que le partage est linéarisable et sans-attente. Le problème résultant est appelé consensus généralisé. Dans ce même article, L. Lamport propose une amélioration de Fast Paxos [Lam06a] pour résoudre le consensus généralisé dans les systèmes répartis par passage de messages. Cet algorithme, appelé Generalized Paxos, résout le consensus généralisé en deux étapes de communication lorsque les accès concurrents sont soit commutatifs, soit spontanément ordonnés par le réseau. Ce résultat est optimal [Lam06b]. Toutefois, si la contention augmente et les processus accèdent de manière concurrente et non-commutative à l'objet partagé, une collision peut avoir lieu. Dans un tel cas, Generalized Paxos nécessite quatre étapes de communication additionnelles, ce qui rend alors l'algorithme moins efficient que Fast Paxos.

Cet article présente *FGGC*, un algorithme permettant de recouvrir d'une collision en une seule étape de communication. *FGGC* emploie un certain nombre d'idées novatrices : (i) nous distinguons les quorums en lecture et les quorums en écriture, (ii) nous introduisons la notion de ronde centrée - une ronde est centrée si tout quorum en écriture attribué à cette ronde contient le coordinateur de la ronde, (iii) quand une collision survient dans une ronde centrée k, nous montrons qu'il est possible de recouvrir en deux étapes de communication si le coordinateur de la ronde k est le coordinateur de la ronde $k+1$. et (iv) nous montrons

ensuite que le recouvrement peut avoir lieu en une étape de communication, si la ronde centrée contient un seul quorum en écriture. De surcroît, Generalized Paxos requiert $3f+1$ processus et en utilise $2f+1$, alors que *FGGC* nécessite $2f+1$ processus et fait usage seulement de $f+1$ d'entre eux. La vitesse et le nombre de pannes franches tolérées par *FGGC* sont ainsi optimaux dans un système réparti par passage de messages. Dans ce qui suit, nous introduisons brièvement le consensus généralisé puis présentons notre solution. Le lecteur intéressé pourra consulter la thèse de P. Sutra [Sut10] pour plus de détails.[†]

2 Le problème du consensus généralisé

Consensus permet un accord sur une séquence de commandes accédant à un objet partagé. Le consensus généralisé adresse le problème de l'accord sur des classes d'équivalence de séquences, appelées structures de commandes. Cette section introduit la notion de structure de commandes, puis énonce le problème du consensus généralisé.

Structure de commande. Soit *Cmd* un ensemble d'opérations, ou *commandes*, sur un objet. Nous notons $(CSeq, \circ, \varepsilon)$ le monoïde libre construit sur *Cmd* ; à savoir : (i) *CSeq* contient toute séquence finie de commandes $\sigma = \langle C_1, \ldots, C_{n\geq 0} \rangle$, (ii) $\circ$ est l'opérateur usuel de concaténation entre deux séquences de commandes, et (iii) ε est la séquence vide. Soit une relation d'équivalence $\sim$ sur *CSeq*. Étant donné une séquence σ, nous notons $[\sigma]$ la classe d'équivalence de σ. L'ensemble des *structures de commande* est le monoïde $(CStruct, \bullet, \bot)$ tel que : (i) *CStruct* est l'ensemble quotient de *CSeq* par $\sim$, (ii) l'opérateur $\bullet$ est défini par : $[\sigma] \bullet [\sigma'] = [\sigma \circ \sigma']$, et (iii) $\bot$ est la classe d'équivalence de ε. Soit $\sqsubseteq$ le pré-ordre réflexif sur *CStruct* induit par l'opérateur $\bullet$, nous supposons ci-après que $\sqsubseteq$ est une relation d'ordre sur *CStruct*. Étant donné un ensemble de c-structs $\mathcal{U}$, nous notons lorsqu'elle existe, $\sqcap\, \mathcal{U}$ (respectivement $\sqcup\, \mathcal{U}$) la borne inférieure (resp. supérieure) de $\mathcal{U}$, et nous supposons que celle-ci est constructible à partir des commandes contenues dans les éléments de $\mathcal{U}$. Par la suite, nous supposons que $\sqcap\, \mathcal{U}$ existe pour tout $\mathcal{U}$. Lorsque $\sqcup\, \mathcal{U}$ existe, nous dirons que $\mathcal{U}$ est *compatible*. Enfin, nous supposons que si pour tout u et v dans $\mathcal{U}$, $\{u,v\}$ est compatible, alors $\mathcal{U}$ est compatible.

Consensus généralisé. Le consensus généralisé est un système réparti constitué de deux ensembles de processus : *Proposeurs* et *Decideurs*. L'état d'un proposeur p est donné par la variable $propose_p$ qui contient l'ensemble des commandes proposées par p au consensus généralisé. Le processus p exécute une seule action : $propose(C)$. Cette action ajoute la commande C au contenu de la variable $propose_p$. Un décideur d maintient une c-struct $decide_d$, et décide une nouvelle c-struct u en exécutant $decide(u)$. Cette action assigne u à $decide_d$. Nous dirons qu'un décideur d a décidé une commande C lorsque $decide_d$ contient la commande C. À l'état initial, pour tout proposeur p, la variable $propose_p$ égale $\{\}$, et pour tout décideur d, $decide_d$ égale $\bot$. Nous supposons ci-après qu'un processus peut subir une panne franche au cours d'une exécution. Si un processus ne subit pas de panne, nous dirons qu'il est *correct*, sinon qu'il est *fautif*. Une exécution du consensus généralisé satisfait les propriétés suivantes :

> **Non-trivialité.** Pour tout décideur d, la c-struct $decide_d$ est toujours constructible à partir des commandes proposées.
>
> **Stabilité.** Pour tout décideur d et pour toute c-struct v, il est toujours vrai que $decide_d = v$ implique $v \sqsubseteq decide_d$ plus tard,
>
> **Cohérence.** L'ensemble $\{decide_d : d \in Decideurs\}$ est toujours compatible.
>
> **Vivacité.** Pour toute commande C et tout décideur d, si d est correct et soit (i) un proposeur correct propose C, ou bien (ii) un décideur décide C, alors d décide C à terme.

3 FGGC

FGGC est un algorithme à la Generalized Paxos [Lam05] pour résoudre le consensus généralisé. Nous considérons dans ce qui suit un système réparti par passage de messages partiellement synchrone dans lequel les liens entre les processus sont quasi-fiables et les processus peuvent subir des pannes franches. *FGGC* utilise quatre ensembles de processus : *Proposeurs*, *Decideurs*, *Coordinateurs* et *Accepteurs*. Nous

[†] Cette thèse présente en particulier une preuve formelle ainsi qu'une évaluation de *FGGC*.

posons f le nombre maximal de pannes franches supportées par *FGGC* pour être vivace, et considérons par la suite l'existence de $2f+1$ accepteurs et de $f+1$ coordinateurs.

Rondes et quorums. *FGGC* exécute une succession de rondes asynchrones. Nous considérons que les rondes sont totalement ordonnées par une relation $<$. Étant donné une ronde k, nous supposons l'existence d'une plus petite ronde plus grande que k, notée $k{+}{+}$. Au cours d'une ronde, chaque décideur tente de décider une ou plusieurs c-structs contenant des commandes proposées. Pour ce faire, *FGGC* s'appuie sur des ensembles d'accepteurs appelées *quorums*. On associe à chaque ronde k un ensemble de quorums en écriture, et un ensemble de quorums en lecture ; nous parlerons par la suite de k-quorum pour désigner un quorum de la ronde k. Une ronde est *régulière* ou bien *rapide*. Elle est liée à un coordinateur unique. Nous supposons ci-après les propriétés suivantes : (**Q1**) Étant donné une ronde k, deux k-quorums en écriture W et W', et un k-quorum en lecture R, il est vrai que : $W \cap W' \neq \{\}$ et $R \cap W \neq \{\}$. (**Q2**) Étant donné une ronde k, deux k-quorums en écriture W et W', et un k-quorum en lecture R, on a que : $W \cap W' \cap R \neq \{\}$. Tout comme dans les précédents algorithmes à la Paxos, les processus de *FGGC* connaissent *a priori* les quorums et le coordinateur de chaque ronde.

Variables et états des processus. Les accepteurs constituent la mémoire stable du système. Ils participent et votent aux différentes rondes. Pour voter durant une ronde, un accepteur doit d'abord participer à cette dernière. L'état d'un accepteur a est constitué de trois variables : la ronde courante à laquelle il participe (rnd_a), la dernière ronde durant laquelle il a voté ($crnd_a$), et la valeur votée au cours de cette ronde ($cval_a$). Au début d'une ronde, le coordinateur tente de convaincre les accepteurs de participer. Si suffisamment d'accepteurs participent, le coordinateur suggère alors aux accepteurs de voter pour une ou plusieurs c-structs. L'état d'un coordinateur c est constitué de deux variables : la dernière ronde commencée ($maxRnd_c$), et la dernière c-struct suggérée par c au cours de cette ronde ($essaie_c$).

Détails de l'algorithme. Une c-struct u est *choisie* à la ronde k, lorsqu'il existe un k-quorum en écriture W, tel que pour tout accepteur a de W, $crnd_a$ égale k et u préfixe la valeur de $cval_a$. Un décideur décide une c-struct u seulement si u est choisie au cours d'une ronde. Une c-struct est *choisissable* à la ronde k si elle est, ou sera ultérieurement, choisie à la ronde k. Une c-struct u est *sûre* à la ronde k lorsque toute c-struct choisissable à une ronde inférieure à k préfixe u. La correction de *FGGC* repose sur l'invariant suivant : (**Sûreté**) Si un accepteur vote pour la c-struct u à la ronde k, alors u est sûre à la ronde k. A partir de cet invariant et l'invariant Q1 ci-dessus, il est facile de se convaincre du fait que *FGGC* assure la clause de cohérence du consensus généralisé. Nous expliquons maintenant comment *FGGC* maintient cet invariant en détaillant le fonctionnement d'une ronde régulière :

- *propose*(C) : Afin de proposer la commande C au consensus généralisé, un proposeur envoie un message propose contenant C à tous les accepteurs et à tous les coordinateurs.
- *phase1A*(k) : Quand le coordinateur c de la ronde k démarre la ronde k, il assigne à $essaie_c$ une valeur nulle, attribue la valeur k à $maxRnd_a$, puis envoie un message 1A étiqueté par k aux accepteurs.
- *phase1B*(k) : Un accepteur a qui appartient à un k-quorum en écriture exécute cette action lorsqu'il reçoit un message 1A étiqueté par la ronde k et que rnd_a est strictement inférieur à k. Il débute alors sa participation à la ronde k en positionnant rnd_a à la valeur k, puis envoie un message 1B étiqueté par k et contenant $crnd_a$ et $cval_a$ à c.
- *phase2Debut*(k,R,l) : Le coordinateur c exécute cette action quand la valeur de $essaie_c$ est nulle, $maxRnd_c$ vaut k, et qu'il existe un quorum R et une ronde l tels que : (i) c a reçu un message 1B étiqueté par k de la part de tous les accepteurs de R, (ii) l est la plus haute ronde pour laquelle un accepteur a précédemment voté dans les messages 1B étiquetés par k et reçus par c, et (iii) pour toute ronde n telle que $l \leq n < k$, R est un n-quorum en lecture. Si ces conditions tiennent, c extrait une c-struct u sûre à la ronde k comme suit : Soit $\mathcal{W}$ l'ensemble des l-quorums en écriture W tels que c a reçu un message 1B de la part de tous les accepteurs de $W \cap R$. Si $\mathcal{W}$ est vide, alors c choisit n'importe quelle c-struct pour laquelle un accepteur a voté lors de la ronde l. Sinon, soit $\gamma(W)$ la fonction qui à un quorum W de $\mathcal{W}$ associe la borne inférieure des c-structs reçus par c de la part des accepteurs de W, le coordinateur c choisit la c-struct suivante : $\sqcup\{\gamma(W) : W \in \mathcal{W}\}$. Cette c-struct est stockée dans la variable $essaie_c$, puis émise dans un message 2A aux accepteurs.
- *phase2AReguliere*(k,C) : Soit C une commande reçue dans un message propose. Lorsque la valeur

de $essaie_c$ n'est pas nulle, et que $maxRnd_c$ égale k, par construction $essaie_c$ est sûre à la ronde k. Dans un tel cas, le coordinateur c concatène la commande C à $essaie_c$, puis suggère $essaie_c$ aux accepteurs dans un message 2A.

- *phase2BReguliere*(k,u) : Quand un accepteur a qui participe à la ronde k reçoit un message 2A contenant une c-struct u, et que $crnd_a$ est inférieur à k, ou que $cval_a$ préfixe u, a vote pour u en assignant u à $cval_a$. L'accepteur a assigne ensuite $crnd_a$ à la valeur k, puis émet un message 2B contenant $cval_a$ à destination des décideurs.
- *decide*(k,W,u) : Quand pour tout accepteur a d'un k-quorum en écriture W, un décideur d reçoit un message 2B de a contenant une c-struct v telle que u préfixe v, le décideur d sait que u est choisie à la ronde k. Dans ce cas d décide u.

Ronde rapide. Lorsque le coordinateur est stable, les accepteurs exécutent une seule ronde. L'algorithme ci-dessus nécessite alors trois étapes de communication pour décider une commande proposée (à savoir la séquence de messages propose, 2A, 2B). Afin de réduire le nombre d'étapes de communication à deux, sa valeur optimal [Lam06b], *FGGC* exécute de surcroît des rondes rapides. Durant une ronde rapide, les accepteurs votent spontanément pour des commandes proposées. Pour ce faire ils exécutent l'action suivante :

- *phase2BRapide*(C) : Un accepteur a exécute cette action quand (i) il participe à une ronde rapide k, (ii) il a accepté une c-struct proposée par le coordinateur, et (iii) il a reçu C dans un message propose. L'accepteur a concatène alors C à $cval_a$, puis envoie $cval_a$ dans un message 2B aux décideurs.

FGGC résout le consensus généralisé en deux étapes de communication si les accepteurs exécutent une seule ronde rapide. Toutefois, du fait de l'asynchronisme et des possibles fautes des processus, les accepteurs peuvent recevoir des commandes non-commutatives dans des ordres différents. Il advient dans ce cas que pour deux accepteurs a et b ayant joint une ronde k, l'ensemble $\{cval_a, cval_b\}$ n'est plus compatible. Un tel évènement est appelé une *collision*. Recouvrir d'une collision nécessite de débuter une nouvelle ronde.

Recouvrement. Pour recouvrir rapidement d'une collision, *FGGC* emploie au choix deux techniques : (**FGGC1**) Nous supposons que tout coordinateur est un accepteur, et par ailleurs que toute ronde rapide k est *centrée*, i.e., que tout k-quorum en écriture contient le coordinateur de la ronde k. Grâce à ces suppositions, quand le coordinateur de la ronde k est le coordinateur de la ronde $k{+}{+}$ et qu'il débute sa participation à la ronde $k{+}{+}$, la c-struct qu'il a acceptée à la ronde k est sûre à la ronde $k{+}{+}$. La première phase de la ronde $k{+}{+}$ est donc désormais locale au coordinateur. Cette technique réduit le temps de recouvrement à deux étapes de communication. (**FGGC2**) Nous réduisons le recouvrement à une seule étape en considérant que (i) les accepteurs envoient les message 2B aux autres accepteurs, et (ii) toute ronde rapide est centrée et contient un seul quorum en écriture. Les accepteurs exécutent l'action suivante lors d'un recouvrement :

- *recouvrir*(u) : Un accepteur a exécute *recouvrir*(u) quand il détecte une collision à la ronde rnd_a et que les conditions suivantes tiennent : (i) rnd_a égale $crnd_a$, (ii) la ronde $rnd_a{+}{+}$ est rapide, (iii) l'accepteur a appartient à un quorum en écriture de la ronde $rnd_a{+}{+}$, et (iv) l'accepteur a a reçu un message 2B de la part du coordinateur de la ronde rnd_a contenant la c-struct u. Dans un tel cas, l'accepteur a participe spontanément la ronde $rnd_a{+}{+}$, et vote durant cette ronde pour la c-struct $\sqcup \{u, \sqcup \mathcal{V}\}$ où $\mathcal{V}$ dénote l'ensemble des c-structs compatibles avec u préfixant $cval_a$.

Références

[Her91] Maurice Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 11(1) :124–149, Janvier 1991.

[Lam05] Leslie Lamport. Generalized consensus and paxos. Technical Report MSR-TR-2005-33, Microsoft, Mars 2005.

[Lam06a] Leslie Lamport. Fast paxos. *Distributed Computing*, 19(2) :79–103, Octobre 2006.

[Lam06b] Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2) :104–125, Octobre 2006.

[Sut10] Pierre Sutra. *Efficient Protocols for Generalized Consensus and Partial Replication*. Thèse de Doctorat, UPMC, Novembre 2010.